Computers Math. Applic. Vol. 25, No. 2, pp. 73–79, 1993
Printed in Great Britain.

0097-4943/93 $5.00 + 0.00

AN INCOMPLETE INVERSE AS A PRECONDITIONER FOR THE CONJUGATE GRADIENT METHOD

JENN-CHING LUO
Department of Civil Engineering and Engineering Mechanics
Columbia University, New York, NY 10027, U.S.A.

(Received March 1992)

Abstract—This paper introduces a new preconditioner with a super convergence for the conjugate gradient method for solving a system of linear equations in a banded structure. The idea is based upon an innovative method of the author to decompose a symmetric matrix into its inverse, and the preconditioner is the result of an incomplete decomposition, i.e., an approximation to the inverse. By the presented preconditioner, the rate of convergence can be significantly improved. Examples to demonstrate the improvement of rate of convergence and the speedup implemented on an Alliant/FX8 computer are included.

1. INTRODUCTION

In scientific and engineering computing, the solution of a system of linear equations is the most important key step. Economical and reliable equation solvers are always necessary for analysis of problems in many disciplines. Once a scientific or engineering problem has been formulated into system equations, an appropriate equation solver is usually necessary. The selection of an appropriate equation solver for application depends on the considered problem as well as on the employed computer. It is very hard and unnecessary to strictly grade the existing methods. Each method usually has favor circumstances, but also has adverse situations; for example, the conjugate gradient method has a high degree of data independence which is suited in a parallel environment, but is very sensitive to the round-off errors. This paper is focused on a preconditioning technique for accelerating the conjugate gradient method for solving a system of linear equations as

$$[A]\{X\} = \{B\}, \tag{1}$$

where $[A]$ is banded, symmetric and positive definite.

Conjugate gradient method [1] is classified into the category of iterative method. In fact, the conjugate gradient method theoretically converges within finite iterations not greater than the number of equations. However, it is very sensitive to the round-off errors, and usually requires more and more iterations to reach a desired accuracy. This disadvantage makes the conjugate gradient method less popular. Modified versions by preconditioners [2–11] have been widely considered in scientific and engineering computing. This work will present another type of preconditioner with a super convergence and an almost perfect speedup, such that the entire procedure, including preconditioner and conjugate gradient method, can take full advantage of multiprocessors.

The presented preconditioner is based upon the author's decomposition [12] to incompletely decompose matrix $[A]$ in Equation (1) into $[C][C]^T$ such that the product of the decomposed matrices is an approximation to the inverse of $[A]$ as $[A]^{-1} \approx [C][C]^T$. Then, $[C][C]^T$ is a preconditioner in this paper, by which the rate of convergence can be significantly improved. Derivations will be shown in the following sections.

The author acknowledges the Advanced Computing Research Facility, Mathematics and Computer Science Division, Argonne National Laboratory, on whose machines the computations of this research were performed.

2. THE AUTHOR'S DECOMPOSITION

The author's decomposition [12] can decompose a symmetric and positive definite matrix into its inverse, for the example of Equation (1) where the matrix $[A]$ can be decomposed into its inverse as

$$[A]^{-1} = [\overline{L}][\overline{L}]^T, \tag{2}$$

where $[\overline{L}]$ is the normalized lower triangular matrix. The procedure [12] for Equation (2) is as follows:

For $j : n \longrightarrow 1$ with step -1, do the following:

(a) For $i : (j+1) \longrightarrow n$ with step 1, do

$$\overline{L}_{ij} \longleftarrow \overline{L}_{ij} * \overline{L}_{ii} + \sum_{k=i+1}^{n} \overline{L}_{ki} * \overline{L}_{kj}; \tag{3}$$

(b) $$\overline{L}_{jj} \longleftarrow \frac{1}{\sqrt{\overline{L}_{jj} - \sum_{k=j+1}^{n} \overline{L}_{kj} * \overline{L}_{kj}}}; \tag{4}$$

(c) For $i : n \longrightarrow (j+1)$ with step -1, do

$$\overline{L}_{ij} \longleftarrow -\left(\overline{L}_{ii} * \overline{L}_{ij} + \sum_{k=j+1}^{i-1} \overline{L}_{ik} * \overline{L}_{kj}\right) * \overline{L}_{jj}, \tag{5}$$

in which each $\overline{L}_{ij}$ shares the computer memory with A_{ij}, and only the lower triangular part of $[A]$ is considered because of symmetry. (A general symmetric matrix [12] can be decomposed into the form of $[A]^{-1} = [L][D][L]^T$). The solution of Equation (1) is then computed by matrix-vector multiplications as

$$\{X\} = [\overline{L}][\overline{L}]^T\{B\}. \tag{6}$$

Eventually, the author's decomposition is also an equation solver which has proved to be highly efficient in a parallel environment [13]. This work will apply the decomposition, but incomplete, to generate a preconditioner for accelerating the conjugate gradient method. Let $[A]$ be a banded, symmetric, and positive definite matrix of order $(n \times n)$, with a half bandwidth m. Equations (3–5) then could be rewritten by the following lemmas where $\overline{L}_{ij}$ shares the computer memory with A_{ij}.

FACT 1. *For column j, $A_{ij} = 0$ where $i > j + m$.*

LEMMA 1. *Equation (3) can be rewritten as*

$$\text{"For } i : (j+1) \rightarrow \min(n, j+m) \text{ with step 1, do}$$
$$\overline{L}_{ij} \leftarrow \overline{L}_{ij} * \overline{L}_{ii} + \sum_{k=i+1}^{\min(n,j+m)} \overline{L}_{ki} * \overline{L}_{kj}." \tag{7}$$

PROOF. (i) In Equation (3) $A_{kj} = \overline{L}_{kj}$ because of sharing with the same computer memory. By Fact 1, $\overline{L}_{kj} = A_{kj} = 0$ where $k > j + m$ such that the upper bound of $\sum$ can be written as $\min(n, j+m)$. (ii) When $i > j + m$, $\overline{L}_{ij} = A_{ij} = 0$ such that the first term in the right side of Equation (3) vanishes; furthermore, the lower bound of k is $i+1$ such that $k \geq i+1$ and $k > j + m + 1 > j + m$ which means $\overline{L}_{kj} = A_{kj} = 0$. This says that when $i > j + m$, the right side of Equation (3) is zero. Therefore, the upper bound of i can be written as $\min(n, j+m)$. This proves the lemma. ∎

LEMMA 2. *Equation (4) can be written as*

$$\overline{L}_{jj} \leftarrow \frac{1}{\sqrt{\overline{L}_{jj} - \sum_{k=j+1}^{\min(n,j+m)} \overline{L}_{kj} * \overline{L}_{kj}}}. \tag{8}$$

PROOF. Since the upper bound of the subscript i in Lemma 1 is $\min(n, j+m)$, $\overline{L}_{kj}$ obtained from Lemma 1 is zero where $k > j+m$. Therefore, the upper bound of $\sum$ can be written as $\min(n, j+m)$. This proves the lemma. ∎

LEMMA 3. *Equation (5) can be modified as*

$$\text{"For } i = n, \min(n, j+m)+1 \text{ with step } -1, \text{ do}$$

$$\overline{L}_{ij} \leftarrow -\left(\sum_{k=j+1}^{\min(i-1,j+m)} \overline{L}_{ik} * \overline{L}_{kj}\right) * \overline{L}_{jj}; \tag{9}$$

$$\text{For } i = \min(n, j+m) \rightarrow (j+1) \text{ with step } -1, \text{ do}$$

$$\overline{L}_{ij} \leftarrow -\left(\overline{L}_{ij} * \overline{L}_{ii} + \sum_{k=j+1}^{\min(i-1,j+m)} \overline{L}_{ik} * \overline{L}_{kj}\right) * \overline{L}_{jj}." \tag{10}$$

PROOF. (i) By Lemma 1, $\overline{L}_{kj} = 0$ where $k > j+m$ such that the upper bound of $\sum$ can be written as $\min(i-1, j+m)$. (ii) By Lemma 1, $\overline{L}_{ij} = 0$ where $i > j+m$. Therefore, the first term in the right side of Equation (5) vanishes when $i > j+m$, which leads to Equation (9). This proves the lemma. ∎

An incomplete procedure for Equations (7)–(10) will be introduced in the following sections as a preconditioner.

3. PRECONDITIONERS

Equations (7)–(10) compute the normalized lower triangular matrix $[\overline{L}]$ such that $[A]^{-1} = [\overline{L}][\overline{L}]^T$. Let $[\overline{L}]$ be partitioned into $[C]$ and $[D]$ such that $[\overline{L}] = [C] + [D]$ in which $[C]$ is in the same banded configuration as $[A]$. For column j, $\overline{L}_{ij}$ where $(i : (j+1) \rightarrow \min(n, j+m))$ defined by Equations (7), (8) and (10) are in the same banded configuration as $[A]$. The matrix $[C]$ then can be defined by Equations (7), (8), and (10) as:

For $j : n \rightarrow 1$ with step -1, do the following:

(a) For $i : (j+1) \rightarrow \min(n, j+m)$ with step 1, do

$$C_{ij} \leftarrow C_{ij} * C_{ii} + \sum_{k=i+1}^{\min(n,j+m)} C_{ki} * C_{kj}; \tag{11}$$

(b) $$C_{jj} \leftarrow \frac{1}{\sqrt{C_{jj} - \sum_{k=j+1}^{\min(n,j+m)} C_{kj} * C_{kj}}}; \tag{12}$$

(c) For $i : \min(n, j+m) \rightarrow (j+1)$ with step -1, do

$$C_{ij} \leftarrow -\left(C_{ij} * C_{ii} + \sum_{k=j+1}^{\min(i-1,j+m)} C_{ik} * C_{kj}\right) * C_{jj}, \tag{13}$$

where each C_{ij} shares the same computer memory with A_{ij}. The matrix $[D]$ is computed by Equation (9), and the inverse of $[A]$ is written as

$$[A]^{-1} = [C][C]^T + [C][D]^T + [D][C]^T + [D][D]^T. \tag{14}$$

Since most coefficients of $[C][D]^T$, $[D][C]^T$, and $[D][D]^T$ are zero, we may use $[C][C]^T$ as an approximation to $[A]^{-1}$ with an error in the form of $[C][D]^T + [D][C]^T + [D][D]^T$. The inexact inverse $[C][C]^T$ can be a preconditioner for accelerating the conjugate gradient method for solving a system of linear equations with a banded, symmetric, and positive definite matrix. The

preconditioned conjugate gradient method then can be written as:

Choose $\{X^0\}$
Set $\{r^0\} = \{B\} - [A]\{X^0\}$
Compute $\{q^0\} = [C][C]^T\{r^0\}$
Set $\{p^0\} = \{q^0\}$
Loop $k = 0, 1, 2, \ldots$
$\alpha_k = \{r^k\}^T\{q^k\}/\{p^k\}^T[A]\{p^k\}$
$\{X^{k+1}\} = \{X^k\} + \alpha_k\{p^k\}$
$\{r^{k+1}\} = \{r^k\} - \alpha_k[A]\{p^k\}$
$\{q^{k+1}\} = [C][C]^T\{r^{k+1}\}$
Test for convergence
$\beta_k = \{r^{k+1}\}^T\{q^{k+1}\}/\{r^k\}^T\{q^k\}$
$\{p^{k+1}\} = \{q^{k+1}\} + \beta_k\{p^k\}$
Endloop.

4. COMPUTATIONAL CONSIDERATIONS

An efficient computing algorithm has to include an economical and reliable numerical method, an efficient data storage scheme, and a clear concept of parallelism. Among these factors, the numerical method plays the most important role. An efficient numerical method has a high potential to be transformed into a parallel algorithm; for example the procedure for computing the preconditioner shown in Equations (11)–(13) may be easily transformed into a parallel procedure by introducing a temporary vector $\{S\}$ of order n as:

For $j : n \longrightarrow 1$ with step -1, do the following:

(a) For $i : (j+1) \longrightarrow \min(n, j+m)$, do independently

$$S_i \longleftarrow C_{ij} * C_{ii} + \sum_{k=i+1}^{\min(n,j+m)} C_{ki} * C_{kj};$$

(b) $$C_{jj} \longleftarrow \frac{1}{\sqrt{C_{jj} - \sum_{k=j+1}^{\min(n,j+m)} S_k * S_k}}$$

where the summation can be computed by a parallel inner product;

(c) For $i : (j+1) \longrightarrow \min(n, j+m)$, do independently

$$C_{ij} \longleftarrow -\left(C_{ii} * S_i + \sum_{k=j+1}^{\min(i-1,j+m)} C_{ik} * S_k\right) * C_{jj}.$$

Steps (a) and (c) in the parallel procedure theoretically can be implemented concurrently, and Step (b) can be computed by a parallel inner product. An efficient performance in computing a preconditioner can be expected.

A data storage scheme should have the capacity to reduce the required computer memories as well as the operations. Since most operations in computing matrix $[C]$ are in column-oriented order. A column-oriented data storage scheme is well suited to access the lower triangular part of matrices $[A]$ and $[C]$. There exist two simple data storage schemes for this purpose. The first one is the standard banded storage in which the coefficient A_{ij} $(i \geq j)$ can be easily and directly

coded in a Fortran code in the form of $A(I,J)$ with the declaration of dimension as

DIMENSION A(m,1),

where m is the half bandwidth. By the standard scheme, the lower triangular part of $[A]$ requires $((n-1)*m+n)$ computer memories. In fact, the banded configuration of $[A]$ requires $(n-1)*m+n-m*(m-1)/2$ computer memories only. The standard banded scheme wastes $m*(m-1)/2$ computer memories. If requiring more computer memory is impossible, the lower triangular part of a banded matrix may be economically accessed by the Fictitious Array Technique [14]; for the example of FAT II in which the coefficient A_{ij} also can be easily and directly coded in a Fortran code as:

$$\begin{cases} \text{A(I,J)}, & \text{where } J \le D, \\ \text{AA(I,J)}, & \text{where } J > D; \end{cases}$$

and associated with the proper declarations of dimension as:

```
DIMENSION A(m,1)
DIMENSION AA(n+1:n-m-1,2*n-m+1:2*n-m+1),
```

where $D = n - m + \text{int}((m+1)/2)$ is the inner divide, and $[AA]$ is a nickname of $[A]$ (in fact, $[A]$ and $[AA]$ share the same computer memory). The Fictitious Array Technique requires the exact $(n-1)m + n - m(m-1)/2$ computer memories for the banded configuration.

5. EXAMPLES AND DISCUSSION

An Alliant/FX8 multiprocessor computer was employed to demonstrate performances of the presented method. The Alliant FX machine permits certain constructions within a single program to form multiple, concurrent execution streams. This kind of machine is most suited for algorithms with a clear concept of parallelism.

The first example was a symmetric and positive definite matrix of order (6×6) with $m = 2$ as

$$[A] = \begin{bmatrix} 10 & & & & & \\ 5 & 11 & & & \text{sym.} & \\ 1 & 6 & 12 & & & \\ & 2 & 7 & 13 & & \\ & & 3 & 8 & 14 & \\ & & & 4 & 9 & 15 \end{bmatrix}. \tag{15}$$

By Equations (11)–(13), the lower triangular matrix $[C]$ is

$$[C] = \begin{bmatrix} 0.369 & & & & & \\ -0.213 & 0.362 & & & & \\ 0.082 & -0.220 & 0.356 & & & \\ & 0.046 & -0.217 & 0.347 & & \\ & & 0.017 & -0.226 & 0.341 & \\ & & & 0.043 & -0.205 & 0.258 \end{bmatrix}. \tag{16}$$

Then the accuracy of the incomplete inverse $[C][C]^T$ can be examined by

$$[C][C]^T[A] = \begin{bmatrix} 1.000 & 0.000 & 0.030 & 0.056 & 0.091 & 0.000 \\ 0.000 & 1.001 & -0.068 & -0.108 & -0.157 & 0.067 \\ 0.000 & 0.000 & 1.037 & -0.009 & -0.069 & -0.295 \\ -0.003 & 0.000 & -0.006 & 1.032 & 0.080 & 0.164 \\ -0.007 & -0.128 & 0.000 & -0.003 & 0.993 & -0.012 \\ 0.000 & 0.003 & -0.134 & 0.000 & 0.000 & 1.000 \end{bmatrix}, \tag{17}$$

which is close to $[I]$. The accuracy of the incomplete inverse is sufficient to be a preconditioner.

A pseudo random process was employed to generate symmetric and positive definite matrices $[A]$ for demonstration of the rate of convergence and speedup. Based upon the random-generated $[A]$, the coefficient vector $\{B\}$ is assumed to be $\{B\} = [A]\{X\}$ where each coefficient of $\{X\}$ is a unit value. The iteration procedure was termined when the residuals $\{r\}^T\{r\}$ were less than 10^{-12} where $\{r\} = \{B\} - [A]\{X\}$. First, a series of testing examples were used to compare the performances between the conjugate gradient method and the presented method. Table 1 showed comparisons of the rate of convergence and the CPU time required by the uniprocessor on an Alliant/FX8 computer. From the results shown in the third and fourth columns in Table 1, it can be seen that the rate of convergence may be significantly reduced; for the example with 400 unknowns, the presented method reduced the number of iterations from 517 to 15. The reason for this improvement can be realized from the fact that the preconditioner presented in this paper is an approximation to the inverse of $[A]$. The fifth column in Table 1 is the ratio of the number of iterations required by the plain conjugate gradient method to the one required by the presented method, which indicates an important result that the rate of convergence can be well improved in large scale problems; for example, the rate of convergence is improved from 7.38 in an example of order (100 x 100) to 34.47 in an example of order (400 x 400). This convinces that the presented preconditioner has a potential to solve large scale systems. The required CPU time is also reduced by the presented method.

Table 2 showed a performance in solving a system of linear equations of order (10000 x 10000) with $m = 250$. The presented method took 31 iterations to solve this (10000 x 10000) example with accuracy up to $\{r\}^T\{r\} < 10^{-12}$. The required 31 iterations also convinces that the presented method with an incomplete inverse has a potential to accelerate the conjugate gradient method. The efficient parallelism can be seen from the timing results shown in Table 2 in which the computing time 3248.20 seconds on 1 processor can be significantly reduced to 428.98 seconds on 8 processors, giving a speedup of 7.66 and an efficiency of 95.75%. As comparing with the performances between the conjugate gradient method and the presented method, the presented method not only can improve the rate of convergence but also can provide an almost perfect speedup in a parallel environment. This work is confined to banded, symmetric, and positive definite matrices. We know that many kinds of engineering and scientific problems can be formulated into banded, symmetric, and positive definite systems. Possibilities of the presented method with a super convergence for such kinds of engineering and scientific computing are endless. Applications of the presented method to finite element analysis are under study.

Table 1. Comparisons between the CG method and the presented PCG method on a processor of Alliant/FX8 computer.

Number of Equations	Half Bandwidth	Iterations		Improved Ratio	Time (sec.)	
		CG	PCG		CG	PCG
100	25	118	16	7.38	5.71	2.45
200	40	222	18	12.33	32.32	9.38
300	70	316	17	18.59	111.67	28.60
400	125	517	15	34.47	401.05	85.25

Table 2. Performance in solving a system of linear equations of order (10000 x 10000) with a half bandwidth 250 on an Alliant/FX8 Computer.

Number of Processors	User Time (sec.)	System Time (sec.)	Total Time (sec.)	Speedup	Efficiency (%)
1	3284.17	0.03	3284.20	1.00	100.00
2	1642.64	0.01	1642.65	2.00	100.00
4	838.57	0.01	838.58	3.92	98.00
8	428.98	0.00	428.98	7.66	95.75

REFERENCES

1. G.H. Golub and D.P. O'Leary, Some history of the conjugate gradient and Lanczos algorithms: 1948–1976, *SIAM Review* **31**, 50–102 (1989).
2. I. Fried, A gradient computational procedure for the solution of large problems arising from the finite element discretization method, *International Journal For Numerical Methods in Engineering* **2**, 477–494 (1970).
3. T.J.R. Hughes, J. Winget, I. Levit and T. Tezduyar, New alternating direction procedures in finite element analysis based upon EBE approximate factorizations, In *Computer Methods for Nonlinear Solids and*

Structural Mechanics, (Edited by S.N. Atluri and N. Perrone) AMD–Vol. 4, pp. 75–109, ASME, New York, (1983).

4. T.J.R. Hughes, A. Raefsky, A. Muller, J.M. Winget, and I. Levit, A progress report on EBE solution procedures in solid mechanics, In *Numerical Methods for Nonlinear Problems*, (Edited by C. Taylor *et al.*), Vol. 2, Pineridge Press, Swansea, (1984).
5. Y. Saad, Practical use of polynomial preconditionings for the conjugate gradient method, *SIAM J. Sci. Stat. Comput* **6**, 865–881 (1985).
6. P. Concus, G.H. Golub, and G. Meurant, Block preconditioning for the conjugate gradient method, *SIAM J. Sci. Stat. Compu.* **6**, 220–252 (1985).
7. J.M. Winget and T.J.R. Hughes, Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies, *Computer Methods in Applied Mechanics and Engineering* **52**, 711–815 (1985).
8. P.G. Ducksbury, *Parallel Array Processing*, John Wiley & Sons, New York, (1986).
9. R. Melhem and D. Gannon, Toward efficient implementation of preconditioned conjugate gradient methods on vector supercomputers, *International Journal of Supercomputer Applications* **1**, 70–98 (1987).
10. C. Farhat and L. Crivelli, A general approach to nonlinear FE computations on shared memory multiprocessors, *Comput. Methods In Appl. Mechanics And Engineering* **72**, 153–171 (1989).
11. G. Mahinthakumar, S. Ratnajeevan, and H. Hoole, A parallelized element by element Jacobi conjugate gradients algorithm for field problems and a comparison with other schemes, *Applied Electromagnetics in Materials* **1**, 15–28 (1990).
12. J.-C. Luo, A new class of decomposition for symmetric systems, *Mechanics Research Communications* (1992) (to appear).
13. J.-C. Luo, A note on parallel processing, *Applied Mathematics Letters* **5** (2), 75–78 (1992).
14. J.-C. Luo, Fictitious array techniques for storing and accessing a symmetric matrix, *Computers & Structures* **41**, 843–852 (1991).